CARNEGIE-MELLON UNIVERSITY

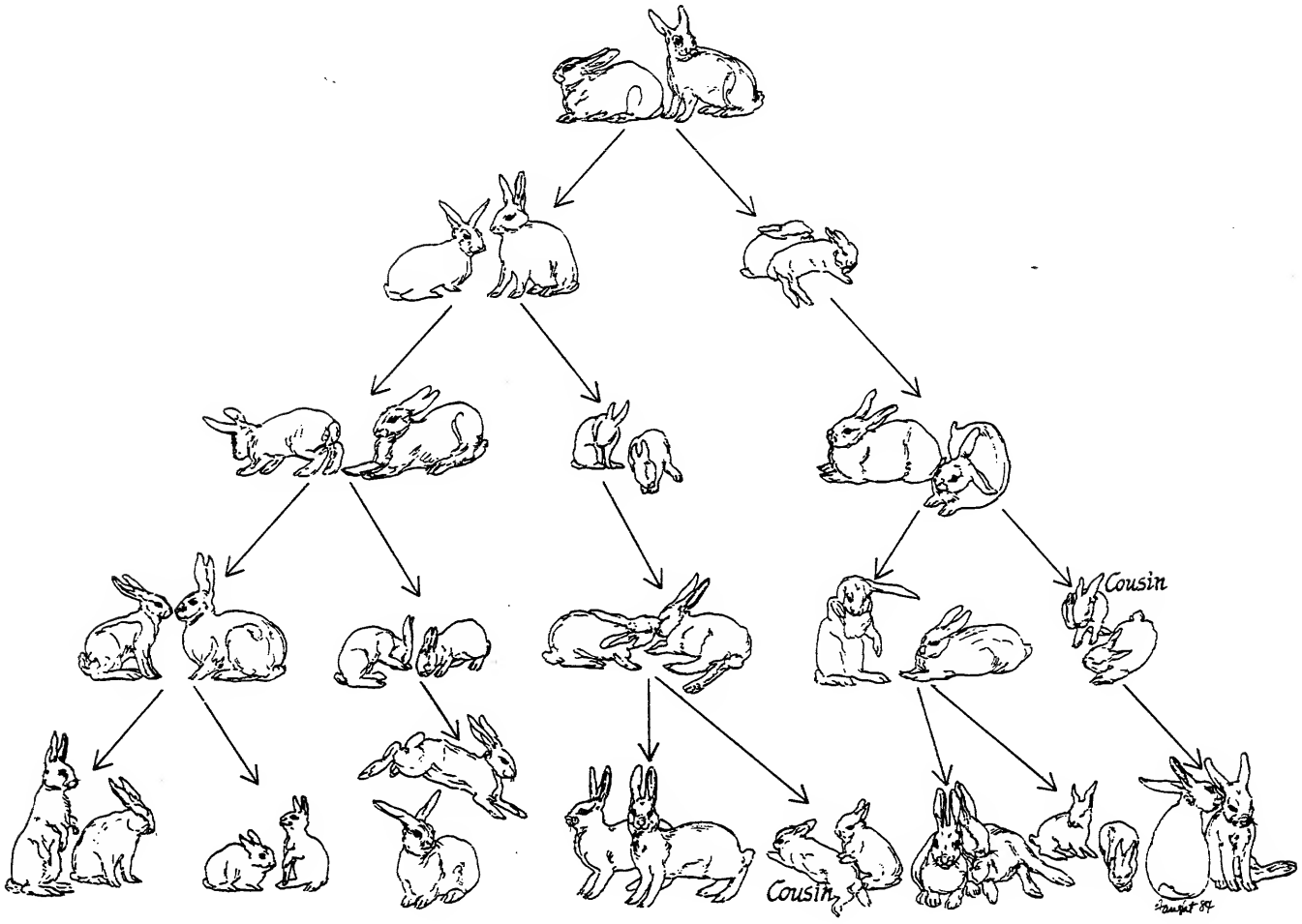
DEPARTMENT OF COMPUTER SCIENCE

SPICE PROJECT

Cousin Users Manual

Philip Hayes Richard Lerner and Pedro Szekely

23 Aug 84



It is easier with a Cousin

23 August 1984
Spice Document S175
Location of machine-readable file: [cfs]/usr/spice/spicedoc/aug84/intro/cousin/cousin.mss

Supported by the Defense Advanced Research Projects Agency, Department of Defense, ARPA Order 3597, monitored by the Air Force Avionics Laboratory under contract F33615-81-K-1539. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Projects Agency or the U.S. Government.

Table of Contents

1 Introduction

COUSIN is a program that provides uniform, cooperative, graphical, command interfaces for a variety of SPICE applications. This manual describes the COUSIN interface system from the viewpoint of the end user of applications which use COUSIN to provide their user interface. If you wish to construct a COUSIN interface to one of your own applications, you will also need the Cousin documentation for application builders which can be found in the SPICE Programmers Manual.

COUSIN interfaces employ a form-based model of communication. Each application has an associated form analogous to the kind of business form in which information is entered by filling in blanks, or circling alternatives. The fields of the form correspond to the various pieces of information that the user and application need to exchange during an interactive session including input parameters, output from the application, and commands to the application. Forms of this kind show the user what his options are and provide a simple yet powerful interface through which COUSIN can provide error detection and correction services.

The end user controls and communicates with a COUSIN application by reading and updating the fields of the form for that application. Since all interaction with forms is conducted via COUSIN, the details of this interaction remain the same for all applications, only the numbers, types, and names of the fields in the forms of the different applications vary. The remainder of this manual describes these interaction details. Information about the forms for specific applications are provided with the descriptions of the applications themselves.

2 Overview of interaction through a Cousin form

Instead of interacting with their user through a typescript, COUSIN applications conduct their interaction through a graphical form. Each application has its own form, and all the forms on a perq are managed by the COUSIN server process. This server process is started automatically in the background without a window when a COUSIN application is run.[1] COUSIN applications are invoked, just like any other SPICE application, by typing their name followed by initial parameter specifications in a shell command line. COUSIN will then prompt you for the position of a window for the form and the shell will return to the command prompt. After start-up, all interaction with the application is conducted through its form, which may be activated, just like any other Sapphire window, by moving the cursor into it and clicking.

In the current version of COUSIN, there are three basic kinds of fields that can be in a form:

- **command fields:** through which the user can issue commands to the application. They look and behave like buttons — one button for each application command.
- **parameter fields:** which contain values that serve as parameters to application commands.

---

[1] This automatic startup involves some time overhead which can be avoided by starting the server (Cousin.Run) yourself in the background before any Cousin applications are run. Eventually, the cousin server will be started when Spice is booted, just like other servers.

- **typescript fields**: through which the application can conduct arbitrary typescript interactions with the user, though they are normally used only for output messages. Note that typescript fields are implemented as separate Sapphire windows, so to type to them it is necessary to make them the listener in the standard Sapphire manner.[2]

Information on what each command or parameter field is for can be obtained by pointing at the field and pressing the Perq "help" key.

A user issues a command to an application by clicking the appropriate command field. The first time a command field is clicked, the parameter fields which provide parameters to *that* command are highlighted by drawing a heavy border around the field. If all the relevant parameter fields are correctly filled, a second click on the command button will cause the command to be issued to the application. On the other hand, if one or more of the relevant parameter fields is incorrectly filled or empty (appropriate feedback will inform the user of these conditions), second and subsequent clicks will have no effect until the user corrects or fills in the offending field(s). After the command has completed, the highlighting is undone, and the parameter fields either revert to their default state (possibly empty) or retain their current values (as specified in the form definition).

There are two standard scenarios for issuing commands, which might tend respectively to be used by experts and novices (relative to the particular application). In the former, the user first edits all parameter fields to be used by the command so that they are correct and complete and then presses the command button twice. In the latter, the user presses the command button once, observes what parameters will be used, makes sure that the values in those fields are all to his liking, and finally presses the command button again. The editing of parameter fields, in either case, is accomplished by direct line editing, value cycling, or menu selection as described in the more detailed descriptions of the interaction facilities provided by COUSIN that follow.

3 Field States

As outlined above, form fields may be in different states at different points in an interaction. This section describes the various states that can occur and what the states depend on. In general, the state of each field is determined by its value and which command field (if any) is currently active. We will treat situations in which a command is active and in which no command is active separately.

---

[2] which, of course, means that the form as a whole is no longer the listener, and must be reselected if you wish to interact with it further.

3.1 No active command

Initially and immediately after the execution of an application command, there is no active command. In this case, all command fields are in the *Neutral* state (thin border) and parameter fields are in one of three states:

Correct:	The field contains the correct type and number of values (thin border)
Empty:	The field contains no value (thin border)
Incorrect:	The field contains an incorrect value (white on black)

When there is no active command, the user can alter or otherwise interact with any of the parameter fields which may change state depending on what is done to them. Clicking a command field will cause that command to become active.

3.2 Active command

There can only be one active command at a time. Clicking a command button when there is no active command will make that command the active one. Clicking a command button when a different command is active will deactivate the previous command and activate the new command. When a command is active, all other command fields are in the *Neutral* state as described above. The active command can be in one of three states, depending on the state of those fields that serve as parameters to that particular command; we will call them the *relevant* parameters.

Ready	All relevant parameters are *correct.* In this state, one more click on the command will cause the command to be issued to the application (thick raised lines)
Bad	One or more relevant parameter fields are *incorrect* or *empty* and must be corrected or filled in before the command can be issued to the application (white on black)
Started	The command has been issued to the application. This state occurs after a click on the field in the *Ready* state, and persists until the application indicates it is ready for more commands. During this time, it is not possible to change the form and any attempt to do so will be ignored (no raised lines)

Note that the only way to issue a command to the application is to click an active command field that is *Ready.* Repeated clicking on *Bad* or *Started* command fields will not do anything (except produce informative messages).

When there is an active command, parameter fields are in one of the following three states:

Correct	The parameter is relevant and correct (thick border)
Incorrect	The parameter is relevant, but is empty or contains an

	incorrect value (white on black)
Irrelevant	The parameter is not relevant (thin border)

4 Interacting with individual form fields

COUSIN provides several ways to interact with the various fields of a form according to their class: command, parameter, or typescript. We call these manipulations COUSIN commands to distinguish them from the commands the user can issue to an application through its COUSIN form. The field to which a COUSIN command is directed is determined by the position of the mouse cursor.

4.1 Command fields

Command fields are displayed as buttons containing the name of the command. There is only one COUSIN command applicable to such fields:

Activate/Start (Any button except WHITE/LEFT):	This command activates a command that is in its *neutral* state, or sends to the the application a command that is *Ready* as described above.

4.2 Parameter fields

A parameter field is displayed in one of two forms distinguished by their placement of values within the field. Packed fields are displayed as a box with the name of the field followed by its one or more values, or in some cases as a box (with differently shaped borders) just containing a single value without a field name. Formatted or Table fields are boxes with their values displayed in columns (in row major order). Table fields may also have a scroll bar at their left which can be used to scroll through the values and also displays the relative location of the entries being shown. The commands which are special to Table fields are described in the section on table fields which follows.

4.2.1 Packed Fields

There are COUSIN commands to change the value of a packed field and to find out about what values can possibly go in it. In those cases where the COUSIN command acts on a particular value in a field, it is necessary to click on the value with the mouse (the YELLOW-BLUE/MIDDLE button) to make it the *current value*. The COUSIN commands for packed parameter fields are:

Display Alternatives (WHITE/LEFT):	For those fields that require their values to be drawn from a fixed set, holding down this button causes a *pull-down* menu to appear, containing the possible values for the field having the current value as an initial substring, allowing the user to choose from it (see Section 7).

DeleteValue
(DEL): This command causes the current value to be deleted.

DefaultValue
(Ctrl-OOPS): Invoking this command causes the current value to be replaced by the default if one is defined.

CycleValue
(GREEN/RIGHT): If the current field has a known universe of values, they can be cycled through using this command. Each time it is invoked the current value is replaced by the next value in the universe of values.

CompleteValue
(INS): If the current field has a known universe of values, this command will cause COUSIN to try to expand the current value. If the completion is ambiguous, the longest initial substring is displayed.

InsertValue
(Carriage Return): This command causes the current value to be given to COUSIN for interpretation and correction, or acceptance. As well as being given explicitly from the keyboard, this command is invoked on a modified value when a different value or field is acted on by the user.

ExpandSubform
(Ctrl->): Values which are representations of other COUSIN forms (subforms) are expanded into their form state through this command. Such values are shown with a thin border around the value.

The user may also edit packed field values directly using a simple line editor with commands which are similar to those of the typescript line editor. In particular, the COUSIN line editor has the notion of an edit cursor which is distinct from the mouse cursor. To set the position of the edit cursor, move the cursor over a value and push a mouse button (the YELLOW-BLUE/MIDDLE button). The edit cursor will be placed at the mouse cursor position. Thereafter, it is moved as expected according to the edit commands given. The implemented commands together with their current key bindings are as follows:

Insert Character
(any alphanumeric character): The character typed is inserted into the string at the current edit position.

Delete character forward
(Ctrl-d): The Character at the edit position is deleted.

Delete character backward
(Ctrl-h, RUBOUT/
BACKSPACE): The character preceding the edit position is deleted.

Delete to end of value (Ctrl-k):	All the characters from the edit position to the end are deleted.
Delete to beginning of value (OOPS):	All the characters before the edit position are deleted.
Move to end of value (Ctrl-e):	the edit position is moved to the end of the value.
Move to beginning of value (Ctrl-a):	the edit position is moved to the beginning of the value.
Move forward (Ctrl-f):	the edit position is moved forward one character.
Move backward (Ctrl-b):	the edit position is moved backward one character.
Set position (YELLOW-BLUE/ MIDDLE):	the edit position is placed at the character closest to the mouse cursor.

4.2.2 Table Fields

The only operation allowed on the values of a table field are Selection, Deselection and Scrolling commands. The usage of these COUSIN commands are:

Select/Deselect One (any button except WHITE/LEFT):	Pressing one of these buttons over an entry in a table will invert the selection of the value.
SelectAll (Ctrl-s):	This command causes all entries in the table to be selected.
DeselectAll (Ctrl-S):	Invoking this command causes all entries in the table to be deselected.
ScrollPageBack (Ctrl-V):	Causes the table to display the previous "page" of values.
ScrollPageForward (Ctrl-v):	Causes the table to display the next "page" of values.
ScrollLineBack (Ctrl-Z):	Causes the table to scroll down one line.

ScrollLineForward (Ctrl-z):	Causes the table to scroll up one line.
ScrollToBeginning (Ctrl-<):	Displays the first line of the table at the top of the table display.
ScrollToEnd (Ctrl->):	Displays the last line of the table at the bottom of the table display.
ScrollLineToTop (YELLOW/MIDDLE):	Pressing this button when over the scroll bar causes the line next to the cursor to be scrolled to the top of the table display.
ScrollTopLineDown (GREEN/RIGHT):	Pressing this button when over the scroll bar causes the top line in the display to be scrolled down to the line next to the cursor.
ScrollToBeginning (any button except WHITE/LEFT):	Pressing one of these buttons when in the header over the scrollbar causes the first line of values to be displayed in the first line of the table display.

In the current release NON-selection is denoted by a line drawn through the value. The user may not edit values in a table field at this time.

4.3 Typescript Fields

There are no COUSIN commands specific to typescript fields. These are standard sapphire windows without titles. Once you have moved the cursor into one and selected it as the listener, you can do anything you could do in an ordinary Sapphire typescript window including scrolling and input line editing. You must reselect the COUSIN Form Window when you are finished in the typescript window.

5 ErrorCorrection, Completion, and Help

Whenever changes are made to a field value, COUSIN checks the value for validity. Whenever a value is determined to be incorrect or ambiguous COUSIN tries to help the user make corrections or clarifications. There are three cases:

Correction:	If the value can be corrected unambiguously then COUSIN will replace the incorrect value with the correct one.
Completion:	If the value is determined to be an initial substring of a

valid value the completed value will replace the incomplete value.

MarkingInvalid: If COUSIN cannot find any single correct choice the value will be marked (by background inversion) as incorrect.

In any case the user is free to make additional changes to the value or go on to something else.

Help on a particular field can be obtained by pressing the HELP key while pointing at the field in question. This will print a short description of the field and its values above the field. For more detailed help there is a Help Application program, and associated form, which can be activated by pressing Ctrl-HELP. This system allows the user to receive help by travelling through a tree of help frames on various topics. The placement of the cursor within the COUSIN form when Ctrl-HELP is pressed determines the root frame. From there, the user selects which frame to see next by selecting, with the mouse, a frame from a set of displayed choices. The first time the Help system is activated there will be a short pause while the system is loaded.

Help and other messages, both from COUSIN and from some applications, are displayed as pop-up messages written in white on black. After one of these messages appear, any keystroke or mouse click will remove the message. There are two COUSIN commands for dealing with these messages:

PreviousMessage (Ctrl-R): This command causes the previous message to be displayed or, if no message is currently displayed, to display the last message displayed.

NextMessage (Ctrl-r): This command causes the next message to be displayed.

Currently the last twenty messages are saved.

To get a listing of all the commands which can be executed from the keyboard, activate the ShowKeyBindings command (Ctrl-?). This will open up a window at the top of your Perq screen and display the key bindings for all of the commands. At this time, the listener is set to this help window and you may scroll it up and down (Ctrl-V / Ctrl-v). To return to COUSIN, press CarriageReturn; COUSIN will not do anything else until you do.[3]

6 Form commands

In addition to the COUSIN commands applicable to fields and values, there are a number of commands applicable to the form as a whole.

RedrawForm (Ctrl-l): causes the form to be redrawn.

RemoveSubForm (Ctrl-<): removes a form which was displayed as a result of an ExpandSubForm

[3] This behavior is a compromise intended to overcome a deficiency in Sapphire and should eventually be remedied.

command.

Abort (Ctrl-c): sends an abort event to the application.

The RedrawForm, Abort and Help commands (Help and Ctrl-?) can be executed even when the form is locked. If the form is locked, all other commands will be ignored.

7 Menus

Pull-down menus are activated by pressing and holding the WHITE/LEFT mouse button over the field for which you want to select a value. When the menu appears under the mouse, you simply move the cursor, still holding the mouse button down, and release the button when you are over the value you wish to select. Moving the cursor outside of the menu and releasing the button will cause no changes to be made to the field. If the values do not fit in the menu, they may be scrolled up or down by moving the cursor to the bottom or top of the menu. For those fields which do not have a set of values to choose from, the message "No menu for this field" will appear while the button is held down.

8 Command Lines, Profiles, and Command Files

8.1 Command lines

Just like other applications in the SPICE system, COUSIN applications can accept initial parameter specifications from command lines. These command lines are processed by COUSIN rather than the application and so are uniform in syntax across all COUSIN applications. A COUSIN command line consists of the command name followed by a sequence of positional parameters, switches with parameters, and switches without parameters. Following the "standard" spice convention, switches are preceded by the slash character ('-'). In quasi-bnf notation:

```
<command-line> ::= <command-name> <parameter>* [&]
<parameter> ::= <token> |                          {positional parameters}
                <switch> [=] <token> |             {switch with parameter}
                <switch>                  {switch without parameter}
<switch> ::= -<identifier>
<token> ::= <identifier> | '<any character but '>*'
```

Whitespace, '-','=', and '&' act as separators.

The names of switches and the number of positional parameters naturally vary from application to application. There is currently no interactive way of finding out what they are for individual applications. It is necessary to look at the documentation for that application. However, when the applications are COUSIN versions of pre-existing SPICE applications, the switches and positional parameters accepted by the COUSIN version will generally be a superset of those accepted by the existing application. For instance, a valid command line for cupdate, the COUSIN version of update, is:

cupdate cousin -host=cad -verbose -retrieve

Also, COUSIN is fairly forgiving about order and spacing, so:

cupdate -host cad cousin -retrieve -verbose

would be interpreted exactly the same way.

All switches and positional parameters in a command line are interpreted as changes to values of fields of the form for the application named at the beginning of the command line. Thus, the above command lines would mean set the `RemoteHost` field of the cupdate form to be `cad`, the `LogicalName` field to be `cousin`, the verbosity field to be `verbose feedback`, and the `Retrieve` command button field to be `on`. These values are inserted in the form before it is displayed. Currently, it is not possible actually to execute application commands from the command line, so specifying command switches, like `Retrieve`, is equivalent to pressing the corresponding command button just once. This limitation will be remedied in future releases.

8.2 Profiles

COUSIN uses the same command line parser to interpret profile files for COUSIN applications. The contents of a profile file for a given application should look like a sequence of command lines for that application without the name of the application at the beginning. In terms of the above bnf-style notation, a profile should contain <parameter>*. Profile lines are interpreted just like the corresponding command line, so a profile for cupdate like:

```
-host=cad -verbose
cousin
-retrieve
```

would have the same effect as our example command line.

When a COUSIN application is invoked, COUSIN first searches for the corresponding profile and interprets that, and only then interprets any command line parameters given with the invocation. Profiles thus may be used to set personalized defaults for application parameters, with the possibility of still overriding those defaults on the invoking command line. A profile file for an application has the same root name as the application and an extension of '.profile'. So the profile file for cupdate would be 'cupdate.profile'. Profiles are looked up along the default search path, so it is best to place them in <boot>.

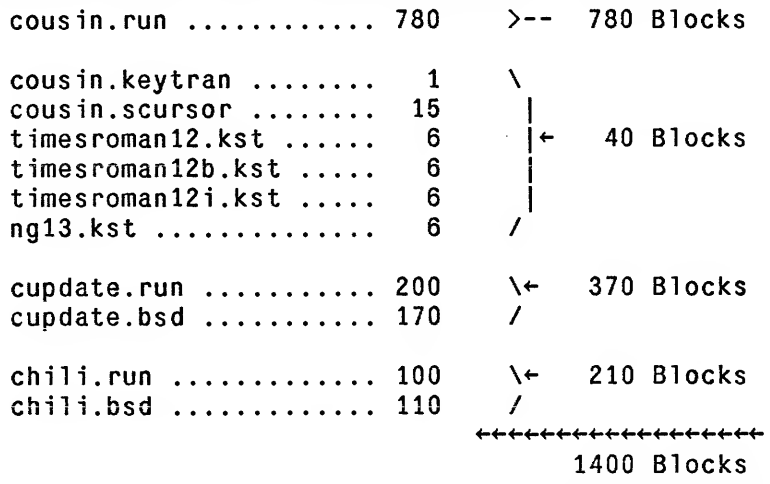
8.3 Command files

When it is possible to execute application commands from command lines, COUSIN will provide interactive facilities to execute command files named by the user. The only difference between these command files and profile files would be their time of execution. Profile files are executed automatically at start up, while command files would be executed interactively as requested by the user after his interaction with the form had commenced.

9 Retrieving and Starting Cousin

COUSIN can be retrieved from CMU-CS-CFS under the logical name COUSIN_a. You may update this logical name into any convenient directory. It will-put the run files for COUSIN and two COUSIN applications, and some .BSD files into your current directory and cousin.keytran, cousin.scursor and some font files into your <boot> directory.

To help you decide where to put COUSIN the approximate sizes of the files are:



The applications which are currently adapted to use the COUSIN interface currently include:

- cUpdate — a graphical interface to the Update program
- Chili — a File Manager

These applications are retrieved along with the COUSIN system, as above. COUSIN applications under development include:

- EnvInterface — a replacement for Show/Define
- ProcMgr U a process manager
- HG — an interface to the Mercury mail system

Please note that these names may change when the applications are released.

COUSIN's Help System requires that the environment variable Cousin-Help: be set to a search list containing the help frame files. This should be rem:Cousin-Help/.

A COUSIN applications are started like any other SPICE applications. When an application starts it will print out a message telling you that it is establishing communication with the COUSIN server. If the COUSIN server has not yet been started, the application initialization sequence will start it.[4] A message in the application window will tell you if this is happening. Once communication with the COUSIN server has

[4] This automatic startup involves some time overhead which can be avoided by starting the server (Cousin.Run) yourself in the background before any **Cousin** applications are run. Eventually, the cousin server will be started when Spice is booted, just like other servers.

been established, the application's window is resized to accommodate the application's form. Once the form has been drawn, you may start interacting with the application through the form.

10 Summary of Cousin commands

10.1 Commands applicable to a form as a whole

Ctrl-l	Redraw Form.
Ctrl-]	Remove SubForm (only applicable to subforms, of course).
Ctrl-c	Send Abort command to application.
Ctrl-?	Show key bindings.
Ctrl-R	Show previous message.
Ctrl-r	Show next message.

10.2 Commands applicable to an individual field

GREEN/RIGHT or YELLOW/LEFT	Activate/Start (for a command field).
HELP	Show short description of field and its values.
Ctrl-HELP	Activate the Help System Form.

10.3 Commands applicable to an individual field value

DEL	Delete a value.
Ctrl-OOPS	Default the value.
Ctrl-p	Replace value with previous value.
GREEN/RIGHT	Cycle the value.
WHITE/LEFT	Pull down a menu of values from which to choose.
INS	Complete a value.
CR	Send a new value to COUSIN for checking.
Ctrl-[	Expand a value to its form representation (if it has one).

10.4 Commands applicable to a table field

Ctrl-V	Scroll to previous page of entries.
Ctrl-v	Scroll to next page of entries.
Ctrl-Z	Scroll down one line of entries.
Ctrl-z	Scroll up one line of entries.
Ctrl-<	Scroll to beginning of entries.
Ctrl->	Scroll to end of entries.
Ctrl-s	Select all entries.
Ctrl-S	Deselect all entries.

10.5 Line editor commands

Any Character	Inserts the character.
Ctrl-d	Deletes current character.
BS/Ctrl-h	Deletes previous character.
Ctrl-k	Delete to end of Value.
OOPS	Deletes to beginning of Value.
Ctrl-f	Move forward one character.
Ctrl-b	Move backward one character.
Ctrl-e	Move to end of Value.
Ctrl-a	Move to beginning of Value.
YELLOW/LEFT, BLUE	Set edit position to mouse position.

Appendix A. The Chili File Manager

A.1 Overview

Chili is a file management utility for Spice. It allows you to browse through the files on your own or other Perqs and delete, copy, rename, or invoke an editor on selected subsets of files. Using Remdef, these facilities are also available for files on a remote Vax. While you can perform destructive file operations using Chili, it contains various safety features to lessen the chance of your doing so inadvertently. Chili's user interface is provided by COUSIN and so is a graphical form. The full set of interface facilities available through COUSIN is described in the COUSIN user's manual which is part of the Introductory Spice Manual. The following description of Chili can be understood without reading the COUSIN manual, but covers only a subset of the features available.

A.2 Obtaining and starting Chili

Chili comes as part of a package of COUSIN applications. This can be obtained from the CFS Vax under the logical name cousin. It will occupy about 1400 blocks all told, but need not go in your boot partition.[5] To start Chili, type 'Chili' to the shell. Depending on circumstances, startup will take between 30 and 90 seconds.

A.3 Browsing

The upper part of the Chili form is concerned with displaying information about files. The **File Spec** field in the upper left hand corner controls what files are displayed. To set the field, click on it with the mouse (middle or right buttons) to make it the *listener* field.[6] You can then type a file specification (using standard Spice syntax including wildcards) into it. Most of the usual line-editing commands (↑a, ↑e, ↑f, ↑d, etc.) apply while you are typing. After you type carriage-return, the files you have specified will be listed in the larger **Files** field immediately below **File Spec.**

The **Display Mode** field to the right of **File Spec** controls the amount of information displayed about each file. It only has two values: **Long** and **Short**. You can type them in as for **File Spec**, or *cycle* them by clicking with the right mouse button. You can also get a menu by *holding down* the left mouse button. If you are still pointing at a menu item when you let up, that value will be placed in the field. Menus and cycling work this way for all fields with a fixed set of possible values.

Especially in **Long** format, it is common for the files display to require more space than is available in the form. The field is therefore scrollable. You can use keyboard commands (↑v, ↑V, ↑z, ↑Z, ↑<, ↑>) while

---

5 A few files (40 blocks out of the 1400) will automatically be placed in your boot partition.

6 The Chili form is a Sapphire window like any other, so if it is not the current Sapphire listener, the first click will merely make it the Sapphire listener and will have no effect on the form itself.

pointing at the field with the mouse, or you can use the scroll bar by the side of the field. The scroll bar works similarly to oil's scroll bar.

The file specification in **File Spec** is interpreted relative to **Current Directory** (at the bottom of the form), so if you want to change "where you're at", type a new directory into that field.

A.4 Manipulating Files

Below the **Files** display, there are command buttons to allow you to **Copy**, **Rename**, **Delete**, or **Edit** files. Commands are invoked by clicking on the corresponding command button twice (using the middle or right mouse buttons). The first click merely highlights the other fields which serve as parameters to the command, thus giving you an opportunity to make sure they are what you really want them to be. The second click actually executes the command.[7]

The primary argument to all Chili's commands is the *selected subset* of files in the **Files** display. After you give a new **File Spec**, all displayed files are selected by default. You can deselect individual files by pointing at them and clicking (middle or right button). Files are marked as deselected by a horizontal line drawn through them. Clicking on a deselected file selects it again. You can also deselect (↑S) or select (↑s) all the files displayed. Selecting or deselecting a file has no effect on the file itself. In particular, deselecting a file does not delete it. You need to use the **Delete** command for that.

For instance, to delete all Oil log files from the current directory, one would type '*+' into **File Spec**, and then press the delete button twice. To clean up a particular Mint file, one might type 'memo.*' into **File Spec**, resulting in memo.doc, memo.mint, memo.mint$, memo.mint+, memo.press being listed in the files display. One would then type ↑S to deselect everything, click on memo.mint$ and memo.mint+ to reselect them, and then press the delete button twice.

The **Copy** and **Rename** commands take an additional destination argument as specified in **Copy/Rename To**. This is a file specification, just like **File Spec**. It may contain wildcards, but they must parallel those in **File Spec**. The interpretation of wildcarded destinations is the same as for the standard Spice copy and rename commands. Like **File Spec**, **Copy/Rename To** is interpreted relative to **Current Directory**.

A.5 Confirmation and other safety features

The **Copy**, **Rename**, and **Delete** commands all use the **Confirmation Request** field to ask for confirmation of each individual file operation. You answer these questions by pressing the **Yes** or **No** buttons to the side of **Confirmation Request**. You can also say "Yes (No), and yes (no) to all further questions generated by this command" by pressing the **All** (**None**) buttons. This confirmation feature allows

---

[7] If any parameters are incorrect or unspecified, they and the command button will turn black on the first click and no subsequent clicks on the command button will have any effect. If that happens, you can correct the problem or go and do something different. If you correct the problem, the black will go away, and the situation will be as though you had clicked the command button once.

you to initiate destructive file operations without having to worry about destroying something you didn't expect to.

If you don't want to be bothered by any confirmation questions, you can cycle (right mouse button) the **Ask for confirmation** field immediately above the answer buttons to **No confirmation**, and Chili will perform all further file operations, destructive or not, without asking you for confirmation. Note that this does not affect an already started **Copy**, **Rename**, or **Delete** command, but only subsequent ones.

If you wish to abort a file manipulation command while it is executing, type ↑Del and then 'c'. This cancels the current command and returns Chili to the neutral state. It does not exit Chili. In fact, the only way to exit Chili is to kill it from a shell. However, in normal use, there is never any reason to exit Chili. If you aren't using it at the moment and don't want it on the screen, iconize it for later use.

A.6 Help and Error Messages

You can get a short explanation of the purpose of any field by pointing at it and pressing the help button on the Perq keyboard. The explanation will appear in a pop-up inverse video message window close to the field you requested help on. Error messages or other informative messages from Chili appear in similar pop-up windows.

All pop-up message windows stay around until you do something else. If you want a message to go away without doing anything else in particular, point to a grey area of the form (outside a field) and click.